Express Mail No. EL727968517US

Docket No. 00 P 7511 US 01

Date of Deposit: March 8, 2001

APPLICATION FOR LETTERS PATENT OF THE UNITED STATES

NAME OF INVENTORS:	BRIAN HENRY STOCKLEY 218 East Chilhowie Avenue Johnson City, TN 37601
TITLE OF INVENTION:	ELECTRONICS ASSEMBLY ENGINEERING SYSTEM EMPLOYING NAMING AND MANIPULATION FUNCTIONS FOR USER DEFINED DATA STRUCTURES IN A DATA SYSTEM USING TRANSACTION SERVICE

TO WHOM IT MAY CONCERN, THE FOLLOWING IS A SPECIFICATION OF THE AFORESAID INVENTION

TITLE OF INVENTION

ELECTRONICS ASSEMBLY ENGINEERING SYSTEM EMPLOYING NAMING AND MANIPULATION FUNCTIONS FOR USER DEFINED DATA STRUCTURES IN A DATA SYSTEM USING TRANSACTION SERVICE

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims benefit, under 35 U.S.C. § 119(e), from U.S. Provisional Patent Application No. 60/188,964, filed March 10, 2000.

BACKGROUND OF THE INVENTION

The present invention generally concerns electronics assembly engineering systems. In particular, the invention concerns methods of permitting naming and manipulation functions for data structures in data systems using transaction service within the environment of an electronics assembly engineering system. In particular, the invention concerns an electronics assembly engineering system having a computer subsystem incorporating computer-readable media for performing the methods and functions of the invention.

Electronics assembly engineering systems for industrial applications require complex and sophisticated subsystems to control performance of each system and to account for changes in the application parameters and the industrial environment. Control has traditionally been provided by computer subsystems. Some computer subsystems use object-oriented databases and transaction service in controlling performance. SIPLACE (by Siemens Electronics Assembly Systems, Inc., 2875 Northwoods Parkway, Norcross, GA 30071 USA), a platform used in the control of the production of printed circuit boards, is an example of an electronics assembly engineering system employing a subsystem using transaction service. SIPLACE employs a programming software subsystem known as SIPLACE Pro, which uses transaction service (in particular, Microsoft Transaction Server) to create and manage data structures, such as objects, within an object-oriented database. Using SIPLACE Pro, programmers can control performance of various aspects of the system through a user-friendly graphical interface. However, in this instance, where the electronics assembly engineering system and computer subsystem employ transaction service and the data structures have referential integrity, programmers working with the system can no longer expect to manipulate the data structures in the same manner as they did in traditional file-based systems.

Microsoft Word is an example of a traditional file-based application working within the file-system of the Microsoft Windows operating system. A user can manipulate files in a directory/file management system, such as Windows Explorer™, or within an editor. Similar results can be achieved in each environment, although the presentation of the functions can be different. When working in the directory/file management system, the file-system is represented by a tree-control. When working in the editor, the user is often presented with a dialog representing the file-system when, for example, "Save As" is chosen from the "file" menu within the Microsoft Word application.

The basic functionality and menu functions of a typical prior art file-based editor are shown in the flowcharts of Figs. 1 and 2. Fig. 1 shows the functionality and menu functions of a prior art file-based editor in the mode where a new file is being created. Fig. 2 shows the functionality and menu functions of a file-based editor in the mode where an existing file is being edited. With reference to Figs. 1 and 2, the menu functions of a traditional file-based editor are characterized as follows:

Manipulating files within the traditional directory/file management system (Prior Art)

Opening a file: A file can be selected in the tree-control and opened either from the context menu or by double clicking (the default function from the context menu). The user's intention is very clear that the file is to be opened in the appropriate editor, i.e. Microsoft Word.

Creating a new file: A new file of a registered type can be created without opening the application by selecting New→<Filetype> from the context menu. This creates the file in the selected folder. The file assumes a default name but is highlighted ready for immediate renaming if the user chooses to.

Renaming a file: The file can be renamed by selecting it and then, once selected, selecting the text of the file name a second time or by selecting it once and choosing the rename function from the context menu. The text of the filename can be edited directly.

Copying a file: A file can be selected and copied by choosing "Copy" from the context menu or from main menu Edit→Copy then using the paste command to create the copy in the desired location within the folder structure. The system is smart enough to rename the file if it is copied to the same folder. The copy function can also be achieved by holding down the control key when a drag is initiated - a copy of the file is created in the folder where the drop is made. The user's intention is clear, a new copy of the file is to be created.

Moving a file: A file can be selected and the "Cut" function chosen from the context menu or main menu Edit→Cut then the paste command can be used to effectively move the file to a new location. The file icon and name remain "grayed out" until "Paste" is initiated. A file can be moved to a different folder by selecting it with a sustained left mouse button and then dragging and dropping it to the new location on the same drive. (If the destination is on a different drive then a copy of the file is made.)

Deleting a file: A file can be deleted by selecting it and then choosing the "Delete" function from the context menu or with the delete keyboard button.

Creating Shortcuts: Selecting the file then choosing the Create shortcut function from the context menus can create a shortcut. Once a shortcut has been made the user can open it, delete it, move it, copy it and rename it in much the same way as if it were the source file. A new shortcut can even be created. The important feature of a shortcut is that when it is opened it is the source object that is opened. A shortcut created from a shortcut is equivalent to a copy of the shortcut – both link directly to the source file. In all other respects a shortcut is an independent file (of a special type) within the file system. A user has no expectation that the source file will be deleted when the shortcut is deleted and, conversely, there is no expectation that the shortcuts pointing to a file will be deleted when the source file is deleted – the shortcuts remain pointing to nothing. Worse than that, if a source file is deleted then later on a new file is created with the same name then the shortcuts will become active again.

<u>Manipulating files within the application (Prior Art)</u>

Open: When "Open" is selected from the File menu a dialog is opened that allows the user to choose an existing file from the file system using a tree control.

New: When "New" is selected from the file menu the user can then select a template for the new document (including the blank document template). The application then creates a file with a default name and puts this in the editor window. This "file" does not really exist in the file-system at this point.

Save: "Save" is typically a background function. The user chooses "Save" from the File menu and the file in the files system is immediately updated with the contents of the editor. The only exception to this is when a file is saved for the first time after it is created which is described below.

Save As: "Save As" is a slightly ambiguous dialog as it covers about three separate functions in a rather clumsy manner.

Save As - Simple - Giving a new file a name: When a file is closed for the first time or when "Save" is chosen for the first time the user is presented with a dialog that shows the default name and the current folder. The user is free to change the text of the name or select another folder.

Save As - To make a copy: If an existing file is in the editor then the Save As dialog can be used to create a copy of the changed file under a different name or in a different folder. In doing this, the original file is left unchanged – the original file and the copy can have different contents at this point.

Save As – To overwrite another existing file: If Save As is selected from an open editor, whether the document is new or existing, it is possible to select an existing file from the file-system and save the document under that name. In this instance, the "Do you want to overwrite the existing file" message appears and the contents of the editor can be saved to the existing file. In this case, the contents of the original file may be unchanged and the overwritten file will have been completely changed. A special case exists when an existing file's own name is selected from this dialog this generally will have the same effect as Save.

Close: A "necessity" of a file-based system is the customary "Do you want to save before closing?" when an editor is closed and changes to the file have not already been saved. Although this could have been avoided in the early days of computer use it is now a standard idiom. When a file is edited it is plain for all to see that there are (at least) two copies of the file – one in memory for editing and one in the file-system. It is reasonable, since there is the possibility of discarding recent changes, to offer the user the choice of discarding the changes and to leave the copy in the file system unchanged. This also leads to the cascading into the "Save As" dialog if the new file has not been named before closing.

The naming and manipulation functions of the traditional file-based systems are not adequate for operating in object-oriented systems using transaction service where data structures have referential integrity. This is especially true when such systems are used in electronics assembly engineering applications. The inadequacy results from several differences between the typical file-system and the object-oriented database using transaction service (such as SIPLACE Pro). These differences lead to quite substantial changes in the overall appearance of the application in the object-oriented system.

- From the user's point of view the object browser and the editor are combined within the client application. The tree controls that allow navigation of the object database are side by side with the editors. The result is that the user does not see the object browser.
- Objects that are opened in an editor are immediately updated when changes are made – there is no copy of the object in memory.
- Objects can have many names or aliases – this leads to a certain amount of ambiguity in that an ordinary user may have trouble visualizing where an object exists. Objects are in a sense omnipresent; it is only their names that have a position in the hierarchy of the tree control.
- The objects within the database have referential integrity – this results in links between objects that have greater significance than their names.

Thus, a host of problems would arise if the basic functionality and menu functions of a typical file-based editor were utilized in an object-oriented database using transaction service. Therefore, it is a goal of the invention to provide naming and manipulation functions for user defined data structures in a data system using transaction service.

SUMMARY OF THE INVENTION

The present invention provides novel naming and manipulation functions for user defined data structures in a data system using transaction service. In particular, in an electronics assembly engineering system having a computer subsystem in which user-defined data structures accessible to editor software have referential integrity and in which user modifications to the data structures during editing are made directly to the data structures rather than indirectly by way of a temporary file, the invention provides a method for permitting naming and manipulation of the data structures. The method includes the steps of (i) providing close, discard and rename functions for the data structures in the case where a newly-created data structure is being edited; (ii) providing close and copy functions for the data structures if an existing data structure is being edited; and (iii) excluding a save-as function for the data structures.

The method is preferably performed by a computer subsystem within the electronics assembly engineering system, and in particular by computer-readable media in the subsystem that includes instructions stored therein for performing the functions. The data structures are typically objects, such as those found in an object-oriented database, but may be other structures, such as mark-up language documents (e.g., XML documents).

The computer subsystem employing the invention is coupled to a display and employs transacted service, wherein the data structures have referential integrity and temporary copies of data structures are not created during editing processes. In accordance with the invention, the display presents a representation of a plurality of data structures and a plurality of functions, including naming and manipulation functions. The plurality of functions displayed will not include a save-as function but do include: close, discard and rename functions if a newly-created data structure is being edited; and close and copy functions if an existing data structure is being edited. The display is preferably a graphical representation of a plurality of data structures and functions.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 shows a flowchart of the basic functionality and menu functions of a prior art file-based editor in the mode where a new file is being created.

Fig. 2 shows a flowchart of the basic functionality and menu functions of a prior art file-based editor in the mode where an existing file is being edited.

Fig. 3 shows an electronics assembly engineering system (having a CPU and data storage) coupled to an human machine interface (HMI) and to an electronics assembly system in accordance with the present invention.

Fig. 4 shows a flowchart of the basic functionality and menu functions of an editor according to the present invention in the mode where a new object is being created.

Fig. 5 shows a flowchart of the basic functionality and menu functions of an editor according to the present invention in the mode where an existing object is being edited.

DETAILED DESCRIPTION OF THE INVENTION

As discussed in the Background above, Figs. 1 and 2 show the basic functionality of a prior art file-based editor. Files at the left side of the figures represent the user selecting the file menu from the main menu of an application. The purpose of Figs. 1 and 2 is to show the ambiguity of the 'Save'/'Save As' concept and how the overall functionality changes depending on whether the file is newly created or is already existing. Furthermore the 'Save'/'Save As' function allows existing files to be overwritten – leading to the reality that 'Save'/'Save As' operates as a copy function.

Figs. 3-5 show the electronics assembly engineering systems and the basic functionality and menu functions of an editor according to the present invention. In the embodiment described hereafter, the data structures used are objects within an object-oriented database. The figures are intended to show how the corresponding functionality of the invention is implemented in a system that uses transaction service. Using the transaction service results in all changes to an object being immediately committed to the database. Consequently, there is no need for a 'Save' function. The functions that are included are inherently modular (as can be seen by reduced cross over between function paths) and, if a function is not applicable under certain circumstances, it is merely disabled. It is critical that 'Save'/'Save As' functionality prevalent in filed-based systems is not used in this application – the substitute functions are really just object name manipulators.

The electronics assembly engineering system employing the invention is shown in Fig. 3. The figure shows the electronics assembly engineering system (having a CPU and data storage) coupled to an human machine interface (HMI) and to an electronics assembly system 10 in accordance with the present invention. The naming and manipulation functions are implemented by computer-readable media stored within the system. The menu functions of the naming and manipulation operations according to present invention are shown in Figs. 4 and 5 and are characterized as follows:

<u>Manipulation of objects within the tree controls</u>

Opening an object: An object can be opened in the appropriate editor by selecting it in the tree and choosing the "Open" function from the context menu or by double clicking on the object name in the tree control. The selected object would be opened in the appropriate editor.

Creating a new object: A new object of a particular type can be created by selecting an object from the tree control and choosing "new object" form the context menu. The new object with default name is created in the same folder and is opened in the appropriate editor. This differs from the file-system method of creating empty files in the same folder and not opening the editor. (The "empty object" is not completely ruled out and may be of some use in certain applications.)

Renaming an object: An object (really a particular alias of an object) can be renamed by selecting the object name in the tree-control and then selecting the text a second time or by selecting the object name and choosing the Rename function from the context menu. The name can be changed by editing the text of the display name directly.

Copying an object: In a file-system the user does not have to make any distinction between the file name and the file – they are the same thing. So when a file is copied it doesn't matter whether the user thinks of this as the file name or the contents of the file. In the systems of the invention, a distinction must be made between an object and the object's name or names. It seems to be fairly clear that if a user of the invention wants to make a copy it is the contents of the object that should be copied not the name but it is the one name of many that the user must select and then choose object copy function. The user would then navigate the tree-control to a particular folder and choose paste. What should the user expect to see now? Should the new object appear in the selected folder with the same name as the object alias that had been selected or should the new object have a system generated default name. This new object would have only one alias regardless of the number of aliases of the source.

Moving an object: There is not much of a problem with selecting an object name and dragging it to another location in the object tree. The sense of moving an object is different from that of the file system. It is only the one name for the object that is being moved. Therefore care must be taken with parallels to cut and past, which would traditionally achieve move in a file-system. This is a strange function because the cut file is in limbo (indicated by a grayed image) until it is pasted, then it disappears from the original location or until another file is copied or cut. To try to imitate cut copy and paste behavior with objects may create some problems.

Deleting an object: An object should be deleted by selecting an object name in the object tree and the choosing the delete option form the context menu or by using the delete keyboard key. As with the previous two examples there is the possibility of ambiguity since the user could be deleting an alias from an object list of aliases or deleting the object itself. The menus would become too cluttered if they provided the user with separate functions for deleting the object and the alias. Since all aliases for a given object are not immediately apparent this calls for a "View aliases" dialog to be created. The "View aliases" dialog would have the capability of adding and deleting aliases for an object.

Creating shortcuts: Creating shortcuts is not possible but multiple aliases or display names will be allowed for each object.

Viewing aliases: The aliases that an object has are not immediately visible since users will be dealing with only one alias for an object at a time. From the tree control it should be possible to select an object name and chose a "View aliases" option from the context menu. This would open a dialog that has a list of all the aliases for this object. New aliases will be able to be added, deleted and possibly edited from this dialog.

Manipulating objects when the editor is open

The following sections represent possible functions of the system of the invention.

Open: An object cannot be "opened" from an open editor. There is no "Open" function in the main menu.

New: An object can be opened for the Main menu Item > New function. The user then is presented with a list of object types from which to choose. The new object is created in a default folder, since there is no context when creating an object in this way, and the new object is presented in the appropriate editor. The new object is given a default name.

Save: The use of the transaction service, and other features of the System, results in changes made to an object in an editor being immediately committed to the database making the Save function unnecessary, to the extent that including a Save function would be ambiguous.

Save As: Similarly the Save As function is also unnecessary, however the additional uses of the Save As function will be addressed separately.

Giving a new object a name: The rename dialog is used to provide the equivalent functionality of a file-system whether the rename function is especially selected or if it appears as the result of closing a newly created object without first renaming it.

Make a copy of an open object: An explicit copy object function is provided in the item menu and allows a new object to be created using the open object as a template. Similar to the file-system Save As operation, this function will open a dialog similar to rename which would allow the user to name and place the new object. The new object will remain in then editor and the original object will be closed.

Overwriting an existing object: It is still a little unclear why, in a system that allows aliases, a user would want to overwrite an existing object resulting in two identical object. Making a new copy of an existing object is clear enough if you consider that the user will then modify the copy, but it is less clear to see an advantage in overwriting. The precedent is that users may be used to doing this in file-based systems.

Close: The editor and object can generally be closed immediately – only when a newly created object is closed is it necessary to prompt the user to rename it.

<u>Solutions within the context of the present invention</u>

With a file-based system, it is clear to the ordinary user that the name of a file, as represented in the file-system, is indistinguishable from the file itself and that shortcuts are merely link files that allow a source file to "appear" at different locations in the file-system. Users believe that the file exists at the particular location in the file-system regardless of the shortcuts. One significant difference with the naming and manipulation functions of the present invention is that the conventional "Save As" functionality is now completely redundant and would in fact be dangerous to attempt to implement because of the expectation that users would carry over from file-based systems. There needs to be new functionality that is obviously different but which allows users to be able to do the sorts of things that they could in a file-system. Hence the creation of the Open as Template function in the tree-control menu and the Copy object in the item menu of the editors.

<u>Functions available in the context menu of the tree controls</u>

Open <object type>: Select object name and choose "Open <object Type> from the context menu. Or double click on an object name. The selected object will open in the appropriate editor.

Open as <object type> template: Select object name and choose "Open as <object type> template. A copy of the selected object will be created with a default name, under the current folder. The new object is opened in the appropriate editor and will be subject to conditions of a new object (see below).

The new <object type>: This option should be available when either a folder or an object is selected. A new object with a default name will be created in the selected folder and will be opened in an editor.

Cut display name: Select object name and choose "Cut display name" from the context menu. The display name is becomes grayed until it is pasted to a new folder location, when the display name is removed from the original location. The user is expected to "paste" the display name into another folder.

Copy <object type>: Select object name and choose "Copy <object type> from the context menu. The user is expected to "paste" the copied object to another location.

Paste: Because of the need to distinguish between copying an object and cutting a display name the "Paste … " option must itself be somewhat context sensitive. If a display name has been cut then "Paste display name" will appear, if an object has been copied then "Paste <object type>" will appear it the context menu.

Paste display name: If a folder is selected and "Paste display name" is chosen from the context menu then the display name that had been copied will now appear in the selected folder. If the display name already exists in that folder for another object then an error message should be generated. This has the end result of moving a display name form one folder to another while retaining the connection to the original object.

Paste <object type>: If a folder is selected and the "Paste<object type> is chosen for the context menu then a new object will be created that is a copy of the source object.

Viewing aliases: Select object name and choose "View aliases" form the context menu. A dialog will open containing a list of all the aliases for the selected object. This dialog is made available for a couple of reasons – to "hide" the aliasing capability from users who may not need to use aliases and to put alias handling in a context outside of the tree controls. For instance there is an ambiguity when a display name is selected and deleted – it could be that the user wants to delete the display name or it could be that the user wants to delete the object. Deleting a display name becomes a feature of the dialog, leaving Delete in the tree control for deleting the object.

Delete object: Select an object and choose "Delete object" from the context menu. The object, for which the selected display name is an alias, will be deleted.

<u>Functions available under main menu "Item"</u>

New object: Select the function under the main menu "Item→New→<object type>". This will create a new object of the selected type with a default name in the default folder (since there is no context) and open the appropriate editor loaded with this new object.

Rename object: Select the function under the main menu "Item→Rename…". The rename dialog will open which will allow a name (alias) to be entered and a folder other than a default folder to be selected. It should be possible to accept the default name and default folder by pressing OK. If a new name is entered or another folder selected than the default name is discarded.

Viewing aliases: Select the function under the main menu, "Item→view aliases…". A dialog will open containing a list of all the aliases for the selected object.

Copy object: This function should only be available for an existing open object or a newly created object that has already been renamed. Select the function under the main menu "Item→Copy object". The "copy object" dialog will open which will allow a name to be entered for the new object and a folder to be selected. The new object will then remain open in the editor but the original object will close. If the original object is not closed the potential exists for many objects to remain open unnecessarily. This function is not available for a newly created object that has not been renamed because that would lead to a cascade of dialogs – one to name the new object and one to name the original.

Close: Select the function form the main menu, "Item→ Close". If the open object is an existing object then the editor window and object should close immediately. If the object has been newly created and the "Rename" function has not been explicitly selected then the "Rename" dialog should open to allow the user to rename it or to accept the default name before the object is closed.

Discard: The "Discard" function has been introduced to simulate the "No: response to the "Do you want to save changes…?" message that appears when a typical file-based editor is closed. Even though the message itself is redundant in the system using the invention it may be useful to allow user to discard a newly created object. Rather than subject the user to an additional question on closing a new object it is better to allow such a function to be explicitly selected. This should only be available for newly created objects that have not yet been renamed and will close the editor and delete the object. To allow the "Discard" function to be available for existing objects that have been re-opened to an editor would be creating an unusual precedent.

It is envisioned that the following additional functions may be provided in certain applications.

Overwrite: It should be possible using the main menu function "Item→Copy object..." to select a display name of an existing object as the name for the "new object. This would have the effect of copying the contents of the open object into the existing object whose display name was selected. This object would then be opened in the editor and the original open object would be closed. An overwrite warning message would be necessary as the original contents of the overwritten object would be lost. It seems that this could also be accomplished by "Copy object" and "Paste object" from the context menu of the tree control, although you have to wonder why, in a system that allows aliasing, you would want to have two identical objects with different sets of aliases. Why not combine the aliases into one object.

Merge: This is an ambiguous relation to the idea of overwriting an existing object with the contents of another. The idea behind a "Merge" type function would be to combine the display names of two different objects resulting in a single object. There is at least one situation where functionality like this may be useful.

Replacing Cut Copy Place functions in the context menu with Copy and Move dialogs: The Cut function is particularly bothersome because there is a sense of the thing that has been cut being in limbo until it is pasted. MS Outlook has a solution to this in that folders in the Outlook tree control cannot be cut and pasted but are instead manipulated with Copy and Move dialogs.